Victorian 6502 User Group Newsletter

# KAOS

For People Who Have Got Smart



OSI SYM KIM AIM UK101 RABBLE 65

Registered by Australia Post
Publication No. VBG4212

ADDRESS ALL CORRESPONDENCE TO 10 FORBES ST., ESSENDON, VIC. 3040

Vol.4 No.12 September 1984

We have just been advised that Compsoft will be closing their Swan St store at the end of September. No information is available about future plans but George assures us he will keep trading and he will let us know what arrangements are being made as soon as possible.

Until new arrangements are finalised Ray Gardiner has asked us to advise anyone with queries about the Rabble 65 to contact him or Jeff Rae. Ray's phone number is Ø58 21 6Ø84 AH and Jeff's is Ø3 741 7963.

The closing date for articles for the October newsletter is Friday 12th October.

The next meeting is on Sunday 3Øth September at 2pm at the Essendon Primary School which is on the corner of Raleigh and Nicholson Streets, Essendon. Would the usual early arrivers please note that the school children will be there from 1Øam to 12 midday, other members are welcome from 1pm.

## FOR SALE

Olivetti DE-523 Terminal, the unit works but has no input/output card. Limited information with the unit. Asking price $65.ØØ
AMDEK DXY-1ØØ plotter, this is just the base with a home made controller. The plotter runs, all that is needed is for someone to write the software. $1ØØ.ØØ
For more info call Nino Benci on

C1P with Tasan Video board (set up as C4) $25Ø.ØØ
8K Tasker Bus RAM board $2Ø.ØØ
Sound generator board - Tasker Bus $1Ø.ØØ
Frank Brown

Superboard II with Rabble expansion board in metal case. 4ØK RAM, power supply built in, alternate character set, 1/2 MHz clock switchable, extended monitor in EPROM, MPI B51 disk drive and controller. Lots of software (tape and disk), manuals and info. $6ØØ.ØØ ono
Ron Kerry

Speech board OSI 48 pin Buss compatible with molex edge connectors. Size 2ØØx1ØØmm. Has SC-Ø1 speech chip, 2732 EPROM with text to speech program plus routines to link ROM BASIC for speech under BASIC control. Also 8 bit parallel port with 4 control lines on board. $13Ø.ØØ
24K static RAM board for 48 pin OSI buss, fully populated. $9Ø.ØØ
Wayne Geary,

## INDEX

SWITCHABLE C4P-C8P
By Bruce Dode

The following mod was devised by Glen Stevens and was carried out on a series II machine, converted to a C4P. It has Rabble and Tasan video boards fitted, dual double-sided 8" drives, George's 'U-Beaut' Alpha-8Ø printer, and runs at 2 MHz. A SPDT switch selects either 4ØK/C4P or 48K/C8P, and a cold start can still be done. The machine has been operating sucessfully (and constantly!) for about 4 months. Requirements are: 3x2732 EPROMs, 4x6116 RAM chips, a SPDT switch, C4PMF and C8PMF Monitor ROMs, and access to an EPROM blower.

A
1. Blow BASIC1 and BASIC 2 in one EPROM, and BASIC 3 and BASIC 4 in another.
2. Blow the C4P and C8P Monitor ROMs in the third EPROM.
3. Fit the switch and earth the middle pin (B).

As the ROM section of the Rabble board cannot be used simultaneously with chips U9 - U13 on the mother board, the switch en(dis)ables BASIC 1&2 and 3&4 on the Rabble board and the upper 2K (C4P) of the Monitor ROM on the mother board, and dis(en)ables the 4 RAM chips on the mother board.

B. RABBLE BOARD MOD

Underneath the Rabble board:-

1. Cut the track between pin 18 of IC37 and earth.
2. Join pin 18 of IC37 via a 3.3K Ohm resistor to the +5V rail.
3. Join pin 18 of IC16 to an outside pin (A) of the switch.

C. MOTHER BOARD MOD

Firstly, carry out 1Ø of the points listed in George's '48K MOD' in KAOS Vol.3 No.1Ø:- Points 1-6, 11 and 13-15. Then:-

(Use pin 18 on the 6116's as the CS line as follows:)

11. Join P1 to pin 18 of U12.
12. Join P2 to pin 18 of U11.
13. Join P3 to pin 18 of U1Ø.
14. Join P4 to pin 18 of U9.

(Use pin 21 of the Monitor ROM to select upper C4P or lower C8P as follows:)

15. Between the +5V and earth rails at the pin 1 end of the Monitor ROM, locate the feed through hole connected to pin 21 of the Monitor ROM.
16. On top of the mother board, cut the link from the feed through to the +5V rail.
17. Under the board, join the feed through via a 3.3K Ohm resistor to the +5V rail.
18. Join the feed through to the third pin (C) of the switch.

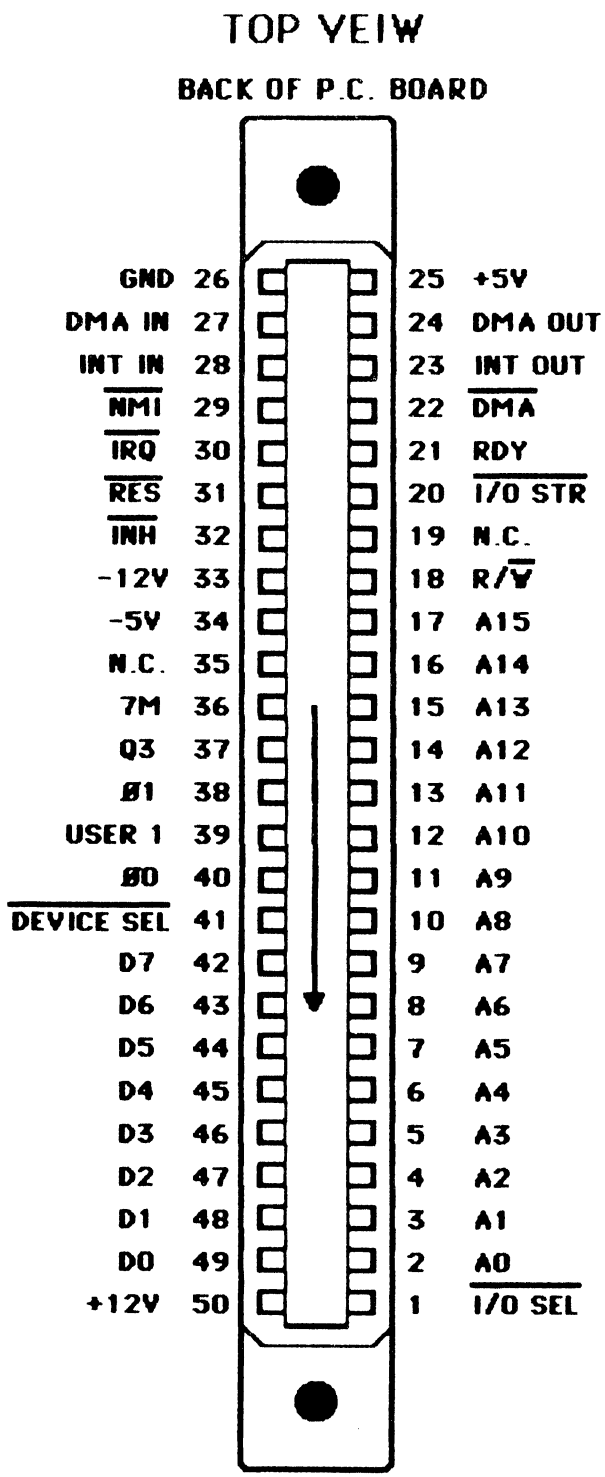
(Use pin 2Ø of U9 - U12 as the CE as follows:)

19. Join pin 2Ø of U9, U1Ø, U11, and U12.
2Ø. Join pin 2Ø of U12 to the feed through in 15.

(NOTE: If the C8P is in the upper half of the Monitor ROM, pin 2Ø of U12 should be connected to pin (A) of the switch.)

21. Insert the Monitor ROM in U13, 6116 chips in U9 - U12, and the BASIC 1&2 and 3&4 ROMs in IC31 and IC28 sockets.

# APPLE INTERFACE SLOTS




PIN#1-I/O SELECT- 256 Addresses are set aside for each peripheral connector. (i.e. C100-C1FF for slot 1): A read or write to such an address will cause this pin on selected connector to go low.

PIN#2 to17-A0-A15- 16 bit system address bus.

PIN#18-R/W- 6502 Read/Write line. High on read, Low on write.

PIN#19-N.C.

PIN#20-I/O STROBE-PIN 20 on all peripheral connectors goes low if addresses $C800-$CFFF are accessed.

PIN#21-RDY-"READY" line to 6502. Line has 3k pullup to +5v

PIN#22-DMA-Direct Memory Access output. Line has 3k pullup to +5v

PIN#23-INT OUT-Interrupt daisy chain output. If not used connect to #28.

PIN#24-DMA OUT-Direct Memory Daisy chain output. If not used connect to #27

PIN#25-+5volts-Positive 5 volt supply; 500ma TOTAL for ALL boards together.

PIN#26-GND-System circuit Ground.

PIN#27-DMA IN-Direct Memory Daisy chain input. If not used connect to #24

PIN#28-INT IN-Interrupt daisy chain input. If not used connect to #23

PIN#29-NMI-Non Maskable Interrupt line from 6502: Line has 3k pullup to +5v: Active low

PIN#30-IRQ- Interrupt request line from 6502: Line has 3k pullup to +5v: Active low

PIN#31-RES-Reset line from "RESET" key on keyboard. Active low.

PIN#32-INH-Inhibit line. When pulled low disables all roms on Mother Board. Line has 3k pullup to +5v.

PIN#33--12volts-Negetive 12 volts. 200ma for ALL boards together.

PIN#34--5v-Negetive 5 volts. 200ma for ALL boards together.

PIN#35-N.C.

PIN#36-7M-Seven Mhz high freq. clock.

PIN#37-Q3-A 2Mhz (nonsymmetrical) general purpose timing signal.

PIN#38-Ø1-Phase 1 clock. Complement of Ø0 clock.

PIN#39-USER 1-Not assigned.

PIN#40-Ø0-Microprocessor Phase Ø0 clock.

PIN#41-DEVICE SELECT-Sixteen addresses are set aside for each peripheral connector. (i.e. $C090-$C09F for slot 1). A read or write to such an address will cause this pin on selected connector to go low.

PIN#42/49-D0-D7-8 bit wide Data bus.

PIN#50-+12v-Positive 12 volts supply. 250 ma Total for ALL boards together.

# Superboard

© Sept. 1984.
Newsletter of the Ohio Superboard User Group, 146 York Street, Nundah, 4012.

## EXPANDING CEGMON'S MENU by Derryl Cocks

My Superboard is mainly used for learning assembly language programming, some wordprocessing, and occasionally writing Basic programs. I'm sure I'll never expand to disk, so for convenience I decided to have an Eprom based OSI. I have a Tasker Buss with an Eprom board containing OSI's Assembler and Extended Monitor, and the WP-6502 word processor.

Neither the assembler nor the word processor will run in Eprom. They have to be first moved down to RAM. There were several possible ways to do this, but finally I decided to expand the menu into the unused disk bootstrap area from $FC00 to $FCA5. CEGMON already has a move routine built into the machine language monitor, and after some disassembling, I discovered how it operated and incorporated it into the routine below. This is lightly commented so you can follow what is being done. The reset routine also needed to be altered and the menu re-written to suit the new choices available. Details of these changes are provided also. I hope this information can be of assistance to members without a disk system. MICRO, February, 1983, contains an interesting article on an Eprom based OSI.

```
FC00 C9 41          CMP #$41    ; is it "A"
FC02 F0 0C          BEQ MVASS
FC04 C9 54          CMP #$54    ; is it "T"
FC06 F0 20          BEQ MVTEXT
FC08 C9 45          CMP #$45    ; is it "E"
FC0A F0 01          BEQ XMON
FC0C 60             RTS         ; None of these, try C W M
FC0D 4C 00 88 XMON  JMP EXMON   ; My Exmon is at $8800
FC10 A9 00    MVASS LDA #$00    ; ASM Eprom first address $7000
FC12 A2 70          LDX #$70
FC14 20 40 FC       JSR STOR1
FC17 A9 FF          LDA #$FF    ; ASM Eprom last address $87FF
FC19 A2 87          LDX #$87
FC1B 20 45 FC       JSR STOR2
FC1E A9 40          LDA #$40    ; Destination address for ASM
FC20 A2 02          LDX #$02
FC22 20 4A FC       JSR STOR3
FC25 4C 40 02       JMP ASMST   ; Start address is $0240 for ASM
FC28 A9 00    MVTEXT LDA #$00   ; WP Eprom first address $9000
FC2A A2 90          LDX #$90
FC2C 20 40 FC       JSR STOR1
FC2F A9 FF          LDA #$FF    ; WP Eprom last address is $87FF
FC31 A2 9F          LDX #$9F
FC33 20 45 FC       JSR STOR2
FC36 A9 22          LDA #$22    ; Destination address for WP code
FC38 A2 02          LDX #$02
FC3A 20 4A FC       JSR STOR3
FC3D 4C 22 02       JMP TEXTST  ; Start address is $0222 for WP
FC40 85 FE    STOR1 STA ST1     ; First address of code to move
FC42 86 FF          STX ST1+1
FC44 60             RTS
FC45 85 F9    STOR2 STA ST2     ; Last address of code to move
FC47 86 FA          STX ST2+1
```

Continued next page

```
FC49 60                RTS
FC4A 85 E4     STOR3   STA ST3     ; Destination address to move to
FC4C 86 E5             STX ST3+1
FC4E 4C E4 FD          JMP MOVE    ; and RTS from there
```

Change from $FF20 to EA EA EA EA 20 ( JSR $FC00 )
" " $FCEA " 20 41 73 6D 2F 45 78 6D 6F 6E 2F 54 65 78 74
A s m / E x m o n / T e x t / C / W / M?

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## SOFTWARE REVIEW - *Animated Lander*

Animated moon lander is a graphics simulation of a lunar lander (yes,another one!) Documentation that comes with this 8k Basic/Machine code program states that it is an accurate simulation, but I fancy the USA would still be trying to make their first successful visit if control was as restricted as in this program! What can I say about a lunar lander program? Everyone must have played one by now.
Well, the graphics are good. There are seven different backgrounds which appear as you attempt to gently lower your craft onto the surface. However you won't get to enjoy them much, because you'll be concentrating on the three gauges at the bottom of the screen. These give height in miles, velocity in MPH, and fuel remaining in gallons. The keys you use to set the burn rate are Ø to 9 (being in hundreds of gallons per five second burn). Once pressed, the burn rate continues until you select another.

The most frustrating part of this game is that by the time you can see the effect of a burn, it is too late to select a new rate. As you get within half a mile, you need a lot of fine tuning, and this is extremely difficult to achieve. Should you make it to the surface, you get a neat presentation of the landing and the raising of the flag.

A full listing for C1 and C4 is provided with the program. Animated Lander is of Progressive Computing (Canada) origins, and I doubt if they are still in business for OSI. The OSUG Library has one at the usual postage rates for library members. Please note that there is only a C1 version on the tape.

NB. Cegmon/Dabug owners should change line 2 to GOSUB 14995 before running, as the I/O vectors are poked to Synmon values for some crazy reason.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## NEW ADDRESS

Victory Software are still selling the two OSI GREATEST HITS packages reviewed in KAOS 3/6 and KAOS 3/8 at US$22.00 incl P&P worldwide. Their new address is : 1410 Russell Road, Paoli, PA 19301, U.S.A.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## M A I L B O X

XENON World imports, PO Box 33, Warradale S.A. 5046, ph (08)296 1033 offer 5¼" SSDD disks in minimum 5 box lots at $139-80 incl freight.

XIDEX disks are offering a special deal for readers of various computer and electronics magazines. Fill in the coupon and selected XIDEX retailers will give you a free box of 5¼ or 8" disks for each box you buy up to a maximum of two. Offer expires October 31st. Neat plastic library case too.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## IMPROVED CHARACTER SET

Mark Howell of offers a character set in a 2716 Eprom which is a direct plug in replacement for the standard OSI one, but offers smaller characters and numbers for better readability, plus a few minor corrections. Cost is $10. Standard ASCII also available at your option.



*Ed Richardson.*

## 6545 VIDEO WITHOUT FLASHING.

B. Wills.

The video board described in KAOS Vol 3, No. 12 and Vol 4, No. 1 has been upgraded to prevent flashing of spurious dots caused by the processor accessing video memory at the same time as the 74LS165 shift register is loaded. This problem also occurs with the original Superboard and I presume with the TASAN board. While not a games addict most of the time, I enjoy Galaxians once in a while. I was disappointed that the flashing during this game made it almost useless with the 6545 video board. Since it is written in machine code, it is able to make very frequent screen updates, and this led to the flashing being particularly noticeable and worse than with the original video. (I suspect that it was worse because the CPU and video clocks were no longer tied to the one master clock).

The Synertek literature describes several methods for screen updates without flashing, but all except one require quite different hardware or hardware plus software. The method now used is called $\emptyset_1/\emptyset_2$ interleave, and requires that video updates occur exactly out of phase to processor R/W activity. The video memory is 'time-shared' - each CPU clock cycle is split so that both CPU and CRTC logic have immediate access. The one constraint (which is significant) is that the video character clock must equal or be an even multiple of the CPU clock. In other respects, hardware is the same as the earlier circuit and no software is required except as before for loading the 6545 registers.

Working backwards, I originally used a 10 MHz dot clock with the 64 char mode, giving a 1.25 MHz character clock. In the new system, the CPU must therefore run at 1.25, 2.5, 5 or higher MHz. My Superboard used to work fine at 2 MHz but seize at 2.5, so it has been set at 1.25 MHz for the new video set-up. This still works with the disc interface (Rabble+Shugart), but I recall when fiddling some time ago that 2 MHz CPU disagreed with the disc. Anyway, 1.25 MHz was acceptable. The snag is with the 24 screen, as in the design it used a 5 MHz dot clock, or 0.625 MHz character clock. This would be unacceptably slow for the CPU - 600 baud cassettes would probably be out, and I guess the disc might object, to say nothing of slow everything else. After several weeks of indecision, I accepted a compromise. I use the 1.25 MHz clock for the 24 screen, so that the display takes 24/64 of the screen width, but with C1 memory map as I tell the 6545 to expect that. In fact, I later made the screen 32 wide, with the display centred, so it's just a cut down 64 width with 64-sized characters running a C1 screen map and software. Galaxians is not as much fun as before as everything is narrower, and its rapid screen usage still doesn't look entirely right, but it doesn't flash. (A type of flicker occurs, but no dots. Other software tested is OK)

For me, the real bonus is with text work using the 64 screen. WP-6502 is less distracting with absolutely no flashing during deleting, scrolling and so on. The general effect is more relaxing to the eye.

### Modifications.

The 74LS73 divider and 74LS157 dot clock switch are omitted, and 10 MHz is fed directly to the 74LS93 as shown. The CPU is disconnected near pin 37 from normal clock track, and fed from the 74LS93 pin 8 on the video board. I made this connection with a piece of unshielded wire-wrap wire about 25 cm long without ill effect. The three 74LS157 video address multiplexers are now switched at pin 1 by a buffered $\emptyset_2$ clock signal taken from the 74LS10 pin 1 or equivalent. Note that the character clock signal from pin 8 74LS93 is no longer inverted prior to feeding to the 74LS20. The ACIA on the Superboard still has original clock connections - no modifications at all.

The Synertek notes show an 8-bit latch (74LS373) to isolate the chargen from the video RAM and buffers, intended to latch the display data if the circuit employs chips with slow access times. Thankfully, the circuit works without this latch, as some butchery would have been needed to fit it. Synertek state that it may be unnecessary depending on chips used. I made no special effort to select chips and all TTL are LS types and the 6116P3 is common enough.

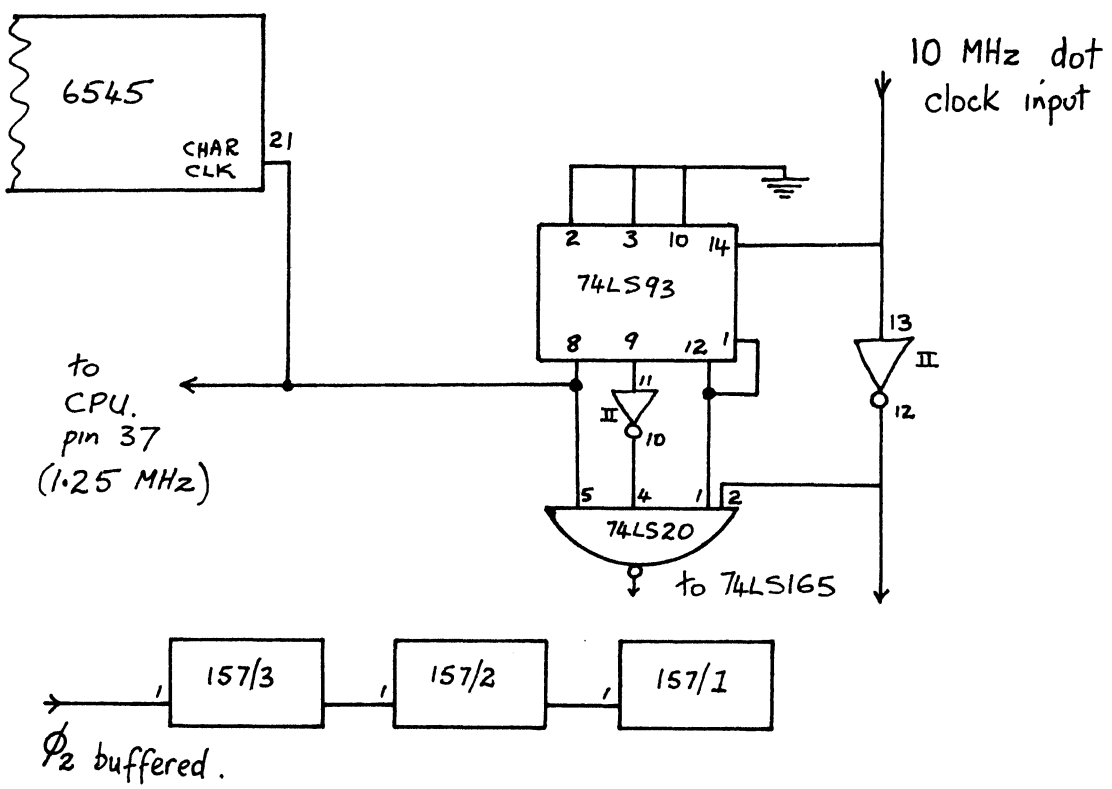
I spent some time experimenting with delays,both in the programmable delay of the 6545, and with extra gates and trial and error capacitors here and there. BUffering the clock going to the CPU with 1 to 4 gates in a 74LS08 (on hand) made very little difference , so no buffering is used. It was necessary to try something as the last character on each line was split, with half of it at the start of the next line but displaced upwards 2 pixels. Intolerable. THe best compromise is as shown, with $\emptyset_2$ being the buffered signal to introduce a bit of delay. I cured the problem finally with a capacitor from the Display Enable signal (pin 18 CRTC) to earth (about 1nF) This sort of fiddling would probably vary a bit between boards if their layout was different, but I cannot comment further as the other board which was built at the time I made mine has not been modified. As a final comment, the earlier articles stated my system was unbuffered, but I made the mods to the Rabble board as per M. Ogden, KAOS Vol 3, No.12 with the addition that ic 2 on the Rabble board was changed from 74LS04 to the open collector version 74LS05. I put 1k pullup resistors on gates in use except the DD line. The system would not work at 2 MHz if the 74LS04 remained, owing to the number of devices connected to DD (Superboard+Rabble+video+extra 2k RAM board).

6545 Registers in same order as previously noted (last 10 zeros not shown)

64: $51 40 48 52 23 0F 1F 21 00 07 (9th parameter = delay = 0)

32: $51 20 39 52 23 0F 1F 23 00 07

If using Cegmon, adjust window accordingly for the 32 line length.



MY SUPERBOARD Part 1Ø
By John Whitehead

Keyboards and Joysticks

This article is partially a re-write of articles by David Anear, Bill Roberts and myself, that have appeared in various KAOS newsletters.

Joysticks for OSI computers consist of five switches, four switches to detect movement of the control up, down, left and right, and a fire button. Forty five degree movement is detected by the closure of two adjacent switches.

You can buy joysticks, or make your own, using David's design which appeared in KAOS Vol.1 No.1Ø. This consists of four micro switches screwed to a plate. The stick is a 1/4" coach bolt about 12Ømm long with about 5Ømm of thread, fitted to a 45mm x 45mm rubber shock mount. (See drawings.)

The series II Superboard is not fitted with a joystick connector so after looking at Bill's article in KAOS Vol.1 No.7, I decided to wire my right joystick up to the W, E, R, T and Y keys, the same as the C4P joystick B. I was not happy using number keys or the N key for the other joystick as accidental movement of the joystick while in the immediate mode could erase BASIC lines or enter NEW. For the left joystick I decided to use the row of keys directly below the ones for the right, that is S, D, F, G and H.

The wiring to the fixed plugs (the socket goes on the joystick leads) is soldered directly to the under-side of the Superboard keys as shown. The wiring to the plug is shown for standard off-the-shelf joysticks.

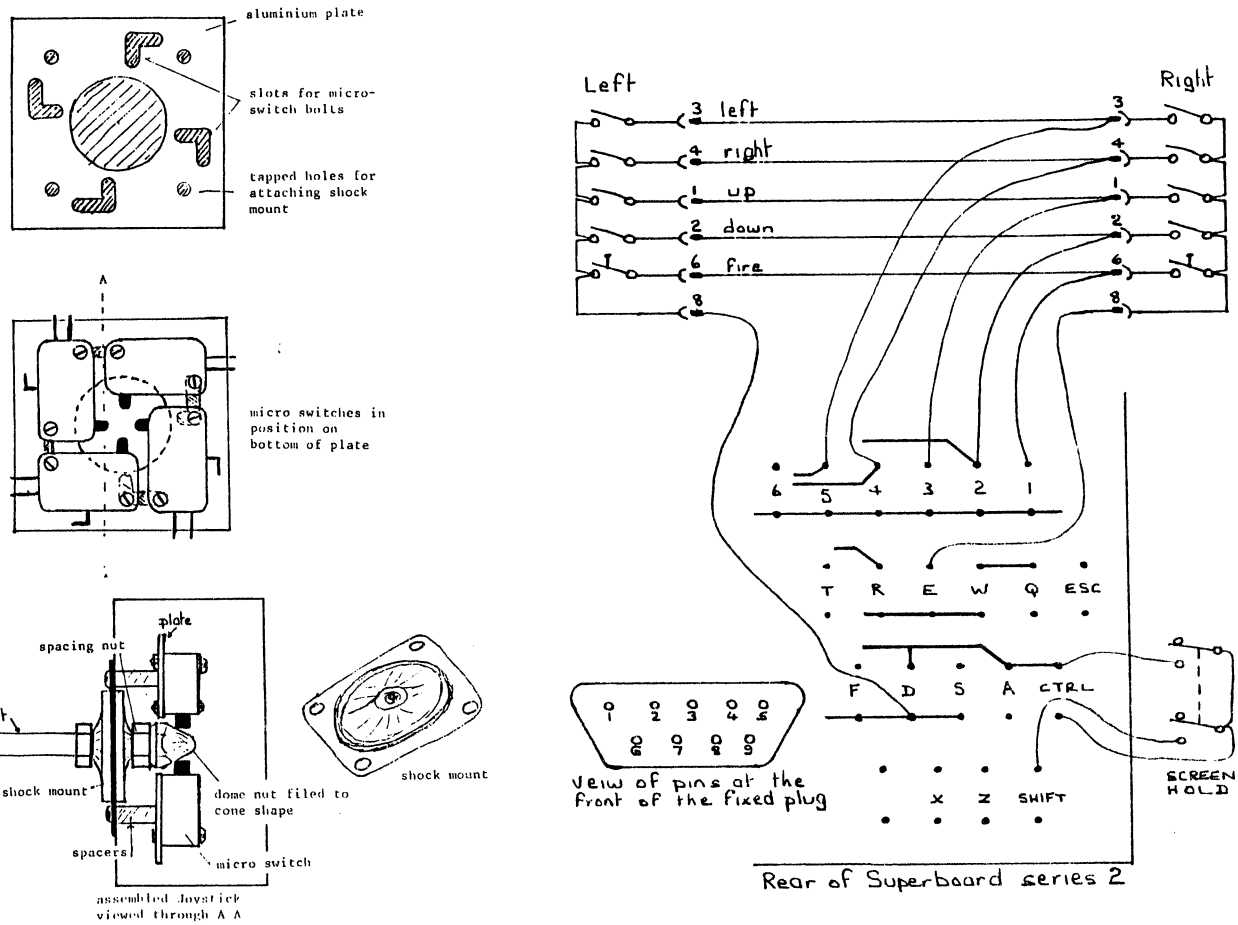
The keyboard wiring drawing also shows a double pole on/off switch. This is a screen hold switch which I have fitted as I am left handed and have difficulty using the usual Dabug screen hold keys.

The keyboard has one address in memory, this is $DFØØ (57Ø88), but it is not fully decoded and actually is $DFØØ to $DFFF. When the keyboard is not being accessed, all the row lines are Logic 1, and when no keys are pressed, all the column lines are Logic 1. When the computer is waiting for ASCII keyboard input it uses a routine at $FDØØ to continously scan the key rows by driving one row low at a time and looking at the columns to see if a key has been pressed, if no key is pressed it drives the next row low etc. The column code found is converted to ASCII and left in the accumulator. BASIC then stores it in the input buffer at $ØØ13 to $ØØ5A. When a BASIC program is running it uses a routine at $FF9B to look for CTRL C by poking rows Ø and 2.

The ASCII keyboard routine is not suitable to detect joystick positions because two or more key closures may need to be detected. A program similar to the one below must be used.

```
1Ø K = 57Ø88: L=247: R=239: REM  For C4 L=8  R=16
2Ø POKE 53Ø,1: REM Turn off CTRL C to prevent false results
3Ø POKE K,L: REM Turn off row 3 for left stick
4Ø LEFT = PEEK(K): REM Get stick position
5Ø POKE K,R: REM Turn off row 4 for right stick
6Ø RIGHT = PEEK(K): REM Get stick position
7Ø IF LEFT = 251 THEN POKE 53Ø,1: STOP: REM For C4 IF LEFT = 4
75 REM If J key is pressed, turn on CTRL C routine and STOP
8Ø PRINT"LEFT =";LEFT,"RIGHT =";RIGHT
85 REM Print joystick position values
9Ø GOTO 3Ø
95 REM For disk, POKE2Ø73,96 = CTRL C off, POKE2Ø73,76 (or 173) = CTRL C on
```

Monster Maze (KAOS Vol 4 No 5) and Close Quarters (KAOS Vol 3 No 12) have been modified for joysticks. They are both on the one cassette and are available from the club library for $4.




## C1P PRODUCES DEBTORS STATEMENTS

by Ken Maclean

For two years I have been using a C1P to produce the Debtors statements for a service station. With about 125 customers, almost 600 debit and credit entries per month, and the operating program, almost 26K of memory is used. Another 3.5K is used during operation.

The system has developed over this period to be completely comprehensive and satisfactory in our situation.
Features of the system are:-

1. Cassette based, simple and accurate.
2. Prints monthly summary and complete debtors list showing, 30, 60 and 90 day details.
3. Incorporates provision to check the integrity of the figures after tape record and re-load.
4. Interest charged on overdue accounts unless customer detail is coded otherwise.
5. 30, 60 and 90 day overdue shown on statement where applicable.
6. Actual statement production - load and run time - 3.5 hours.
7. 210 mm (80 col) paper cut in photo trimmer. 140 mm part is folded twice, then into window envelope. 70 mm part is our copy.
8. No statement produced for inactive accounts.

John Whitehead has a tape with the program and sample data to illustrate the system. Any one interested can contact me for further information.
Ken Maclean,

UNDERSTANDING FORTH Part 3

*By Ray Gardiner*

Forth control structures are the subject for this month. We will examine the following words and present examples of their usage.

Loops are among the most common structures in programs; Forth provides for several types.

THE FINITE LOOP (loops through a fixed number of times).

```
DO .... LOOP
DO ....+LOOP
```

THE INFINITE LOOP (loops through indefinitely).

```
BEGIN .... UNTIL
BEGIN .... AGAIN
BEGIN .... WHILE .... REPEAT
```

PROGRAM BRANCHING STRUCTURES (CONDITIONAL)

```
IF .... THEN
IF .... ELSE .... THEN
CASE .... OF .... ENDOF .... ENDCASE
```

UNCONDITIONAL BRANCHES LIKE "GOTO" ARE NOT ENCOURAGED.

It is possible to add your own specialised control structures as required by the application.

A set of control structure primitives has been derived by Kim Harris and these will be described later. But now let's look at the standard looping and branching words and see what they do

```
DO .... LOOP   (end+1 start --- ....) increments by +1
DO .... +LOOP  (end+1 start --- ....) increment from stack
I              ( .... --- index)      copy current index
LEAVE          ( .... --- ....)       force a DO LOOP to terminate
BEGIN .... <flag> UNTIL               if flag is false loop back
                                      if flag is true, exit loop
BEGIN ....<flag> WHILE .... REPEAT    loop while flag is true
                                      'repeat' is unconditional loop
                                      back to begin
```

: DO LOOP

When the word DO is encountered while compiling, the primitive (DO) is compiled into the definition.

The word (DO) simply moves the top two items on the parameter stack to the return stack where they are used by LOOP to count down the number of times the loop should be executed. the top of the return stack is called the loop INDEX and the second element on the return stack is called the LIMIT. At any point within the loop the INDEX may be retrieved by simply copying the top of the return stack to the parameter stack, the word for this is I, which is functionally identical to R that we discussed in part two.

```
: COUNTUP  100 0 DO I . LOOP ;
```

When the word COUNTUP is executed the two numbers on the parameter stack are moved to the return stack, the index initially is zero and the limit is 100, I copies the index to the parameter stack and "dot" outputs the number in ASCII to the terminal when we hit LOOP which increments the INDEX and compares it to the LIMIT, if they are not equal then control branches back to immediately following the DO.

The nett effect of all this is that the words between DO and LOOP are repeated the number of times specified by the difference between the two numbers on the stack.

+LOOP behaves in a similiar fashion to LOOP but instead of incrementing the limit by +1 each time as LOOP does, +LOOP takes its increment from the stack, this way you can loop forwards or backwards by making the increment positive or negative.

NOTE: Since (DO) takes the INDEX and LIMIT from the parameter stack there is no reason why the INDEX and LIMIT need to appear within the definition at all. They can be supplied from outside the definition just as easily.

```
: STAR ." *" ;   (every system needs a word to print *'s).
```

Supplying the limit inside the definition

```
: 5STARS 5 Ø DO STAR LOOP ;
```

In the example below the limit is not specified so we may by supply it at run time.

```
: STARS Ø DO STAR LOOP ;
   1ØØ STARS   (will print 1ØØ stars)
```

The loop index and limit can take any value as the following definition shows. Can you explain how it works?

```
: CHARS  53Ø4 53248 DO I 53248 - I C! LOOP ;
```

: INFINITE LOOPS

As there is no means to decide in advance how many times the loop will execute the indefinite loop is probably a better name. Loop termination depends on a conditional test which returns a flag that is tested to decide whether or not to keep looping. We will now study the simplest construction of this type.

BEGIN .... UNTIL

BEGIN simply marks the target of the backward branch at compile time, UNTIL is a conditional branch to the words immediately after BEGIN. The top of the parameter stack is used to pass the flag to UNTIL . If the flag is [false]=Ø then control is passed back to just after BEGIN, if the flag is [true] ie. some non-zero value then control passes down to whatever follows the UNTIL.

Most Forth systems have a word which scans the keyboard and returns a flag if the <break> (^C or whatever depending on the system) key is pressed. This word is usually called ?TERMINAL and returns a false flag if the <break> key is not down, and a true flag if it is.

```
: DEMO BEGIN
          CR ." Infinite loops are fun "
          ?TERMINAL
       UNTIL  ;
```

: BY THE WAY You should attempt to layout your definitions with control structures indented and clearly identified. Nice indenting and commenting goes a long way to making bug finding easier and makes complex code more readable.

\ Control structures may be nested to any depth and combined to perform more complex tasks.

\ The following word dumps memory in any base from any address, checking after every line to see if the operator wants to continue.

```
: MEMDUMP                          \ starting address on the stack
      BEGIN                        \ start dumping
           CR                      \ next line
           16 Ø                    \ 16 bytes /line
              DO                   \ do one line
                 DUP I + C@ 3 .R   \ do one location
              LOOP                 \ back for next
           16 +                    \ next address
           ?TERMINAL               \ check operator
      UNTIL                        \ loop back to CR
   DROP CR ;                       \ tidy up as you leave FILE
```

: CONDITIONAL BRANCHING STRUCTURES

The IF .... THEN structure is the simplest of these structures, IF compiles as a conditional branch to just after the THEN. If the top of the stack is true then the branch is not taken and the words between IF and THEN are executed.

As and example suppose we wish to define a word which will output ASCII characters to the video display, prefixing any non-printing characters with "^" and masking out any non-ASCII characters.

Range ØØ - 31 character will output as ^A, ^C, etc.
Range 32 - 127 character will be output as normal ASCII

```
: ASCII.EMIT  127 AND DUP  32 < IF ." ^" 64 + THEN EMIT ;
Ø2 ASCII.EMIT   ^B
```

To test this definition completly we could construct a simple test loop as follows.

```
: TEST.ASCII.EMIT 256 Ø DO I ASCII.EMIT LOOP ;
```

: COMPARISON TESTS

The following are presented with stack effects using the notation introduced in the last section.

| Forth word | Stack effect | Comments |
|---|---|---|
| < | n1 n2 --- flag | true if n1 less than n2 |
| > | n1 n2 --- flag | true if n1 greater than n2 |
| = | n1 n2 --- flag | true if n1 is equal to n2 |
| Ø< | n --- flag | true if n is negative |
| Ø= | n --- flag | true if n is zero |

It should be noted at this point that FORTH-83, the new standard, has redefined the value for the true flag to be -1 rather than just non-zero. In most applications the change will mean no alteration to existing code, however in cases where arithmetic operations are performed on flags, the code will require alteration to use the new standard.
The change really means that for a flag to be true all bits must be set to 1, ie $FFFF = -1, rather than just $ØØØ1.

To test if a number is within a range, say 3Ø to 4Ø, two tests will be required.

```
: ?RANGE DUP 3Ø < IF ." less than 3Ø "   THEN
         DUP 4Ø > if ." greater than 4Ø " THEN
         DROP ;
```

CONDITIONAL BRANCHING

To test the definition, we can write another definition to run through the range of values.

```
: ?TEST.RANGE  6Ø 2Ø DO CR I ?RANGE LOOP ;
```

In some applications it is necessary to specify action to be taken if a test is false, not just the true case.

```
<flag>  IF  [flag was true]
        ELSE  [flag was false]
        THEN
```

```
: ?RANGE  3Ø < IF    ." less than 3Ø "
              ELSE   ." greater than or equal to 3Ø "
              THEN   ." always print this "
```

Like looping structures, conditional branching may be nested up to any depth. However the special case of nested IF ELSE THEN structures has been generalised by Dr. Eaker and has been adopted as a standard CASE statement, it must be noted that an infinite number of alternate CASE (see note**) statements can exist and the one presented here is representative of current trends in Forth.
Note** The source for the CASE statement will be given next month.

Suppose you wish to test the keyboard and depending on the value returned execute some particular words.

```
: ACTION KEY CASE  ASCII A      OF ." you pressed A"       ENDOF
                   ASCII Z      OF ." you pressed Z"       ENDOF
                   CONTROL C    OF ." goodbye folks" QUIT ENDOF
             ENDCASE ;

: TEST.ACTION BEGIN ACTION AGAIN ;
```

A complex command interpreter or menu selection scheme can be constructed around the case statement with relative ease.
Some early Forth systems may not have the words ASCII and CONTROL, if you wish to add these to your system here are the definitions I am currently using.

```
: ASCII 32 WORD HERE 1+ C@
         STATE @ IF (compiling)  [COMPILE] LITERAL
                 THEN ;   IMMEDIATE

: CONTROL 32 WORD HERE 1+ C@ - [ 32 255 XOR ] LITERAL AND
         STATE @ IF (compiling) [COMPILE] LITERAL
                    THEN ;   IMMEDIATE
```

NOTE: Control converts to upper case ie. CONTROL c = CONTROL C.

Finally conditional tests can be constructed within looping structures.
Suppose we want a word that will loop a fixed number of times unless the operator intervenes and stops the program.
The word LEAVE is defined to simply set the index equal to the limit. A possible definition of LEAVE could be:

```
: LEAVE  R> DROP R> DUP >R >R ;
```

In practice LEAVE is written in assembler, however the above definition would work exactly the same, albeit slower.

```
: STARS  Ø DO STAR  ?TERMINAL IF LEAVE THEN LOOP ;
```

1ØØØØ STARS <cr> . While that's running hit the ^C key to stop.

EXERCISES

1. Write a Forth word to dump memory in hex and ASCII, a page at a time, or until the operator presses ^C. If a full page has been dumped, wait until the operator presses the space bar, then dump another page, again checking for ^C.

ANSWERS TO PART 2 (STACK MANIPULATION)
1. (i) Enter the number [a] on the stack

```
DUP DUP DUP * * K1 * SWAP DUP DUP * K2 * SWAP K3 * K4 + + +
```

(ii)

```
DUP ROT DUP ROT SWAP + ROT SWAP - /
```

2.

```
?WIN SWAP 6 * + ROT ROT SWAP 6 * + SWAP - ;
```

Apologies for the cramped presentation of the answers, but space is at a premium this month.

## USING 5.25" and 8" DISK DRIVES
By Jeff Rae

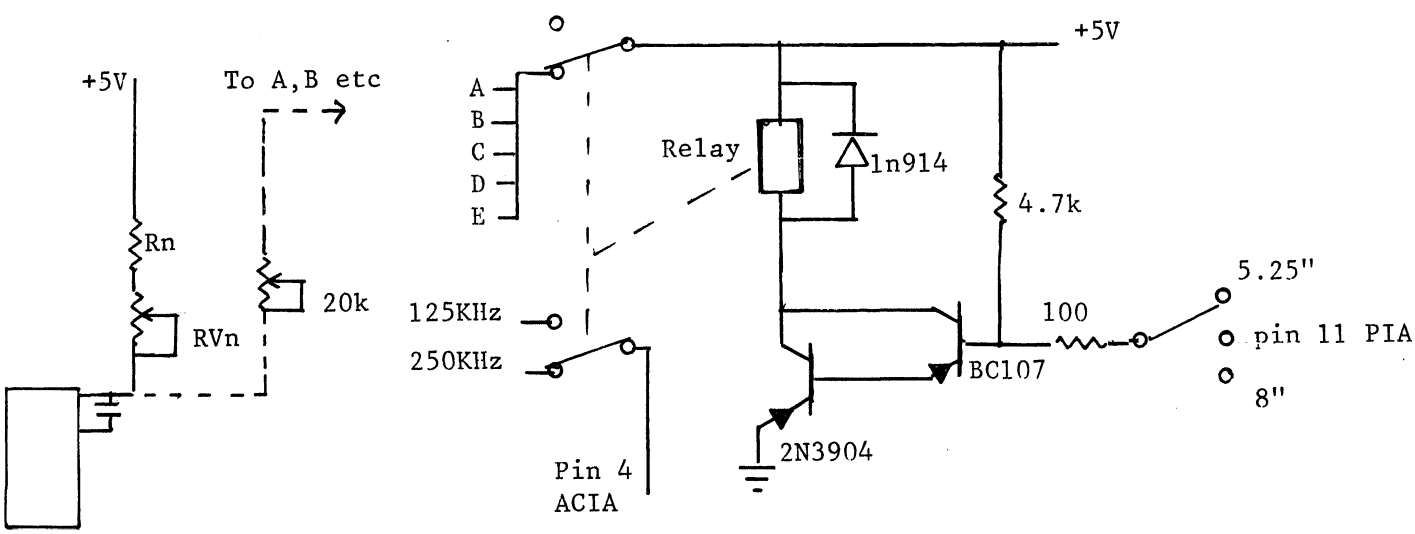
There was a query in the last KAOS newsletter about using a 5.25" as the A drive and an 8" as the B drive. Yes, it can be done. I have a pair of 8" drives and a pair of 5.25" drives connected to the same disk controller. But because the start of workspace for the two versions of DOS is different it is not possible to load a program from one format and expect it to run on the other. However Ohio in their wisdom, built into the DOS a feature that allows you to tranfer BASIC or assembler file from one to the other. To use this feature you have to do two POKEs to set up the indirect file. You then list your source file into the indirect file, close this indirect file, and switch your system to boot the other format. Do your POKEs to setup the indirect file then use (CTRL X) to bring your source file back into the current workspace.

If you are unsure on the use of indirect files, I suggest you look for the relevant information in one of the reference manual as a full explanation is beyond the scope of this article, however I have included the schematics for the hardware modifications below.

The timing circuit shown is typical of the five timing signals that have to be changed. These signals are Tx Clock (R3 & RV1), Tx Data (R4 & RV2), Rx Data (R6 & RV4) and two signals on the data separator consisting of (R12 & RV5) and (R14 & RV6). All values in brackets refer to components on the Rabble Expansion board. I have also placed a three position switch in the circuit. Two positions on this switch allow you to physically select which drive is the A drive, and the third position allows software selection.

You will also have to make up an interconnection board to connect the 36 and 50 way cables. You will have to work out these connections yourself as there are some differences between the different makes of drives.

If you have any problems you can ring me on




---

KAOS Queensland meeting, 5/8/84.Attendance:-10 Computers:-4
The meeting opened at 12 noon. John Froggatt and Robin Wells were first in with the BBCs. The software count had grown considerably since the last meeting, and we were entertained with a feast of new games. John removed the case to show us his new Sir Rom board which allowed expansion to the maximum 256k Rom.

Alan Calvert had his C1/4 with doublesided 8 inch drive and treated us to a demo of the features including the M/C Eprom programme he had written to



Continued next page

work with the Bernie Wills Rabble Board Eprom programmer. The machine was not yet fully debugged, and the current problem was crashing disks.

The only other machine was a standard C1P. A couple of the members wanted to know if the DTACH board mentioned in the last KAOS was available for the OSI/RABBLE. No details were known.

The next meeting of the QLD group will be on October 7th.

Registered by Australia Post
Publication No. VBG4212

If undeliverable return to
KAOS, 10 Forbes St
Essendon, Victoria 3040

KAOSKAOS
K Postage K
A Paid A
O Essendon O
S 3040 S
KAOSKAOS